Hybrid Global-Local Indexing for Efficient Peer-to-Peer Information Retrieval

Chunqiang Tang and Sandhya Dwarkadas
Computer Science Department, University of Rochester
{sarrmor,sandhya}@cs.rochester.edu

Abstract

Content-based full-text search still remains a particularly challenging problem in peer-to-peer (P2P) systems. Traditionally, there have been two index partitioning structures—partitioning based on the document space or partitioning based on keywords. The former requires search of every node in the system to answer a query whereas the latter transmits a large amount of data when processing multi-term queries. In this paper, we propose *eSearch*—a P2P keyword search system based on a novel hybrid indexing structure. In eSearch, each node is responsible for certain terms. Given a document, eSearch uses a modern information retrieval algorithm to select a small number of top (important) terms in the document and publishes the *complete* term list for the document to nodes responsible for those top terms. This *selective* replication of term lists allows a multi-term query to proceed local to the nodes responsible for query terms. We also propose automatic query expansion to alleviate the degradation of quality of search results due to the selective replication, overlay *source* multicast to reduce the cost of disseminating term lists, and techniques to balance term list distribution across nodes.

eSearch is scalable and efficient, and obtains search results as good as state-of-the-art centralized systems. Despite the use of replication, eSearch actually consumes less bandwidth than systems based on keyword partitioning when publishing metadata for a document. During a retrieval operation, it searches only a small number of nodes and typically transmits a small amount of data (3.3KB) that is independent of the size of the corpus and grows slowly (logarithmically) with the number of nodes in the system. eSearch's efficiency comes at a modest storage cost, 6.8 times that of systems based on keyword partitioning. This cost can be further reduced by adopting index compression or pruning techniques.

1 Introduction

Peer-to-Peer (P2P) systems have gained tremendous interest from both the user and research community in the past several years. First-generation systems such as Gnutella and KaZaA are already prevalent, and second-generation systems such as PAST [32] and CFS [14] based on Distributed Hash Tables (DHTs) are under serious development (e.g., the IRIS project [19]). With a gigantic amount of information in these systems, it would be impossible for users to remember or even know the place or precise ID of the desired data. The capability to retrieve documents using content-based full-text search would greatly improve the usability of these systems.

Although it is possible to build a dedicated search engine to index contents in P2P systems in a way similar to how Google indexes the Web, a P2P search system built on top of the same nodes already used in the P2P storage system is particularly attractive because of its low cost, ease of deployment, availability, and scalability. In this paper, we study the challenging problem of building P2P keyword search systems.

To facilitate the retrieval of documents, a distributed (not necessarily P2P) search system places information regarding the occurrence of terms (words or phrases) in documents in the form of *metadata* at certain places in the system. The metadata placement strategy in existing systems are based on either *local* or *global* indexing [42].

In local indexing (see Figure 1 (i)), metadata are partitioned based on the document space. The complete term list of a document is stored on a node. A term list $X \rightarrow a, c$ means that document *X* contains terms *a* and *c*. During a retrieval operation, the query is broadcast to all nodes. Since a node has the complete term list for documents that it is responsible for, it can compute the relevance between the query and its documents without consulting others. The drawback, however, is that every node is involved in processing every query, rendering systems of this type unscalable. Gnutella and search engines such as AllTheWeb (www.alltheweb.com) [30] are based on variants of local indexing.

In global indexing (see Figure 1 (ii)), metadata are distributed based on terms. Each node stores the complete inverted list of some terms. An inverted list $a \rightarrow X, Z$ indicates that term *a* appears in document *X* and *Z*. To answer a query consisting of multiple terms, the query is sent to nodes responsible for those terms. Their inverted lists are transmitted over the network so that a join to identify documents that contain multiple query terms can be performed. The communication cost for a join grows proportionally with the length of the inverted lists, i.e., the size of the corpus. Most recent proposals for P2P keyword search [16, 20, 28, 39] are based on global indexing, but with enhancements to reduce the communication cost, for instance, using Bloom filters to summarize the inverted lists or incrementally transmitting the inverted lists and terminating early if sufficient results have already been obtained. In the following, we will simply refer to these systems as *Global-P2P* systems.

Challenging conventional wisdom that uses either local or global indexing, we propose a *hybrid indexing* structure to combine their benefits while avoiding their limitations. The basic tenet of our approach is *selective* metadata replication. Like global indexing, hybrid indexing distributes metadata based on terms (see Figure 1 (iii)). Each node j is responsible for the inverted list of some term t. In addition, for each document D in the inverted list for term t, node j also stores the complete term list for document D. Given a multi-term query, the query is sent to nodes responsible for those terms. Each of these nodes then does a local search without consulting others, since it has the complete term list for documents in its inverted list. Our system based on hybrid indexing is called *eSearch*. It uses a Distributed Hash Table (DHT) to map a term to a node where the inverted list for the term is stored. We chose Chord [38] for eSearch, but other DHTs such as Pastry, CAN, and Tapestry can also be used without major changes to our design.

When naively implemented, eSearch's search efficiency obviously comes at the expense of publishing more metadata, requiring more communication and storage. We propose several optimizations that reduce communication by up to 97% and storage by up to 90%, compared with a naive hybrid indexing. Below we outline one important optimization—top term selection.

Each document contains many words. Some of them are central to ideas described in the document while the majority are just auxiliary words. Modern statistical information retrieval (IR) algorithms such as vector space model (VSM) [33, 36] assign a weight to each term in a document. Terms central to a document are automatically identified by a heavy weight. In eSearch, we only publish the term list for a document to nodes responsible for top (important) terms in that document. Figure 1 (iv) illustrates this optimization. Document X contains terms a and c but its term list is only published to computer 3 since only term c is important in X.

This optimization, however, may degrade the quality of search results. A query on a term that is not among the top terms of a document cannot find this document.

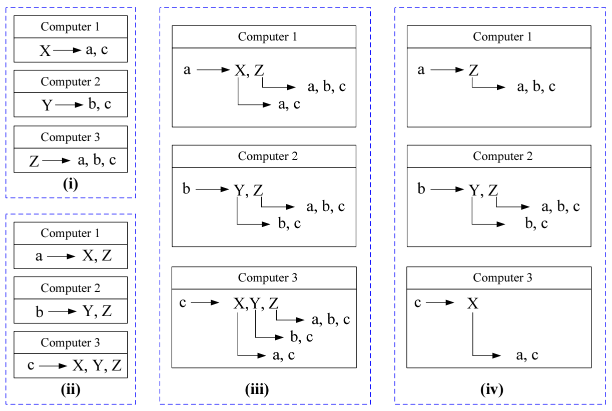

Figure 1: Comparison of distributed indexing structures. **(i)** Gnutella-like local indexing. **(ii)** Global indexing. **(iii)** Hybrid indexing. **(iv)** Optimized hybrid indexing. *a*, *b*, and *c* are terms. *X*, *Y*, and *Z* are documents. This example distributes metadata for three documents (*X*-*Z*) that contain terms from a small vocabulary (*a*-*c*) to three computers (1-3). Term list $X \rightarrow a, c$ means that document *X* contains term *a* and *c*. Inverted list $a \rightarrow X, Z$ indicates that term *a* appears in document *X* and *Z*.

Our argument is that, if none of the query terms is among the top terms for a document, IR algorithms are unlikely to rank this document among the best matching documents for this query anyway. Thus, the top search results for this query are unlikely to be affected by skipping this document. In Section 3, we quantify the precision degradation due to this optimization. Our results show that eSearch obtains search quality as good as the centralized baseline by publishing a document under its top 20 terms.

In order to further reduce the chance of missing relevant documents, we adopt automatic query expansion [23]. We draw on the observation that, with more terms in a query, it is more likely that a document relevant to this query is published under at least one of the query terms. This scheme automatically identifies additional terms relevant to a query and also searches nodes responsible for those terms. We also propose an overlay *source* multicast protocol to efficiently disseminate term lists, and two decentralized techniques to balance the distribution of term lists across nodes.

We evaluate eSearch through simulations and analysis. The results show that, owing to the optimization techniques, eSearch is scalable and efficient, and obtains search results as good as the centralized baseline. Despite the use of metadata replication, eSearch actually consumes less bandwidth than the Global-P2P systems when publishing a document. During a retrieval operation, eSearch typically transmits 3.3KB of data. These costs are independent of the size of the corpus and grow slowly (logarithmically) with the number of nodes in the system. eSearch's efficiency comes at a modest storage cost (6.8 times that of the Global-P2P systems), which can be further reduced by adopting index compression [43] or pruning [7].

Given the quickly increasing capacity and decreasing price of disks, we believe trading modest disk space for communication and precision is a proper design choice for P2P systems. According to Blake and Rodrigues [2], in 15 years, disk capacity increased by 8000-fold while bandwidth for an end user increased by only 50-fold. Moreover, work in the Farsite project [3] observed that about 50% of disk space on desktops was not in use.

In this paper, our focus is on designing an indexing architecture to support efficient P2P search. Under this architecture, we currently use Okapi [31, 36] (a state-of-the-art content-based IR algorithm) to rank documents. In reality, search engines combine many IR techniques to rank documents. The adoption and evaluation of those techniques in our architecture is a subject of future work.

The remainder of the paper is organized as follows. Section 2 provides an overview of eSearch's system architecture. Sections 3 to 5 describe and evaluate individual pieces of our techniques, including top term selection and automatic query expansion (Section 3), overlay source multicast (Section 4), and balancing term list distribution (Section 5). We analyze eSearch's system resource usage and compare it with Global-P2P systems in Section 6. Related work is discussed in Section 7. Section 8 concludes the paper.

2 System Architecture

Figure 2 depicts eSearch's system architecture. A large number of computers are organized into a structured overlay network (Chord [38]) to offer IR service. Nodes in the overlay collectively form an eSearch *Engine*. Inside the Engine, nodes have completely homogeneous functions. A client (e.g., node X) intending to use eSearch connects to any Engine node (e.g., node E) to publish documents or submit queries. Engine nodes are also user nodes that can initiate document publishing or a search on behalf of their user.

Among nodes in the P2P system, we intentionally distinguish server-like Engine nodes that are stable and have good Internet connectivity from the rest. Excluding ephemeral nodes from the Engine avoids unnecessary maintenance operations. Moreover, not even every stable node needs to be included in the Engine, so long as the Engine has enough capacity to offer the desired level of quality of service.

When a new node joins the P2P system, it starts as a client. After being stable for a threshold time (e.g., 20 minutes) and if the load inside the Engine is high, it joins the Engine to take over some load, using the protocol described in Section 5. Data stored on an Engine node are replicated on its neighbors. Should a node fail, one of its neighbors will take over its job seamlessly.

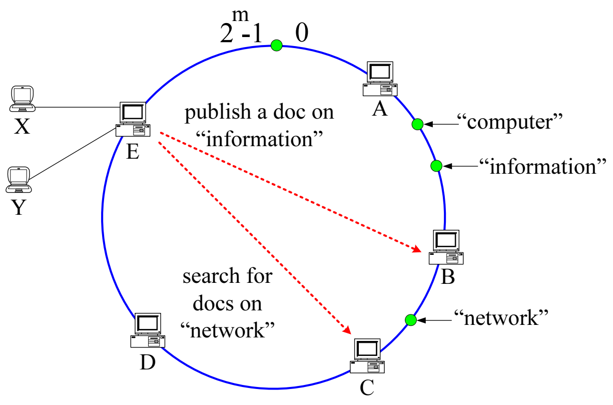

Figure 2: System architecture of eSearch.

Inside the Engine, nodes are organized into a ring topology corresponding to an ID space ranging from 0 to $2^m - 1$ where $m = 160$. Each node is assigned an ID drawn from this ID space, and is responsible for the key range between its ID and the ID of the previous node on the ring. Each term is hashed into a key in the ID space. The node whose ID immediately follows the term's key is responsible for the term. For instance, node B is responsible for inverted lists for the term "computer" and "information". Lookup for the node responsible for a given key is done through routing in the overlay. With the help of additional links not shown in Figure 1, Chord on average routes a message to its destination in $O(\log N)$ hops, where N is the number of nodes in the overlay.

Document metadata are organized based on the hybrid indexing structure. To publish a document, a client sends the document to the Engine node that it connects to. The Engine node identifies top (important) terms in the document and disseminates its term list to nodes responsible for those top terms, using overlay source multicast to economize on network bandwidth (see Section 4).

A client X starts a search by submitting a query to the Engine node E that it connects to, which then uses overlay routing to forward the query to nodes responsible for terms in the query. Those nodes do a local search, identifying a small number of best matching documents, and return the ID and relevance score (a numerical value that specifies relevance between a document and a query) of those documents to node E. Node E gives returned documents a global rank based on the relevance score and presents the top documents to client X. To improve search quality, node E may expand the query with more relevant terms learned from returned documents, start a second round of search, and then present the final results to client X (see Section 3.2).

To avoid processing the same query repeatedly and also to alleviate hot spots corresponding to popular queries, query results are cached for a certain amount of time at the nodes processing the query and the paths along which the query is forwarded. If a query arrives at a node with live cached results, those results are returned immediately.

3 Top Term Selection and Automatic Query Expansion

In Section 1, we proposed hybrid indexing for efficient P2P search, by combining the advantages of local and global indexing while avoiding their limitations. Like global indexing, it distributes metadata based on terms. Like local indexing, it stores the complete term list for a document on a node. Given a query, it searches only a small number of nodes and avoids transmitting inverted lists over the network, thereby supporting efficient search. The drawback, however, is that it publishes more metadata, requiring more storage and potentially more communication.

Our first optimization is to avoid publishing a document under *stopwords*—words that are too common to have real effect on differentiating one document from another, e.g., the word "the". This simple optimization results in great savings since a significant portion of a document's content could be stopwords. Our second optimization is to use vector space model (VSM) [36] to identify top (important) terms in a document, and only publish its term list to nodes responsible for those terms.

3.1 Top Term Selection

We first provide an overview of VSM. VSM assigns a weight to each term in a document or query. Terms central to a document are automatically identified by a heavy weight. VSM computes the relevance between a document D and a query Q as

$$\text{relevance}(D, Q) = \sum_{t \in D,Q} d_t \cdot q_t \tag{1}$$

where t is a term appearing in both document D and query Q, d_t is term t's weight in document D, and q_t is term t's weight in query Q. Documents with the highest relevance score are returned as search results. The weight of a term is decided by several factors, including the length of the document, the frequency of the term in the document, and the frequency of the term in other documents. Intuitively, if a term appears in a document with a high frequency, there is a good chance that the term could be used to differentiate the document from others. However, if the term also appears in many other documents, its importance should be penalized.

A myriad of term weighting schemes have been proposed, among which Okapi [31, 36] has been shown to be particularly effective. For instance, among the eight systems that achieved the best performance in the TREC-8 *ad hoc* track [41], five of them were based on Okapi. We adopt Okapi in eSearch but omit its details here due to space limitations. Okapi relies on some global statistics (e.g., the popularity of terms) to compute term weights. Previous work [17] has shown that statistical IR algorithms can work well with estimated statistics. eSearch uses a combining tree to sample documents, merge statistics, and disseminate the combined statistics. In the following, we assume the global statistics are known. An evaluation of eSearch's sensitivity to the estimated statistics is the subject of ongoing work.

We conduct experiments on volumes 4 & 5 of the TREC corpus [41] to determine if it is true that some terms in a document are much more important than others. The TREC corpus is a standard benchmark widely used in the IR community. It comes with a set of carefully constructed queries and manually selected relevant documents for each query, against which one can quantitatively evaluate the search quality of a system. It includes 528,543 documents from the news, magazines, congressional records, etc. The average length of a document is 3,778 bytes.

Cornell's SMART system [5] implements a framework for VSM. We extend it with an implementation of Okapi and use it to index the TREC corpus. SMART comes with a list of 571 stopwords, which is used as is in our experiments. The SMART stemmer is used without modification to strip word endings (i.e., "book" and "books" are treated as the same).

For each document, we sort its terms by decreasing Okapi weight and compute the relative weight of each term to the biggest term weight in the document. We average this normalized term weight across all documents, computing a mean for each term rank, and report these means in Figure 3 (a). Note that the Y axis is in log scale. The normalized term weight decreases exponentially as the term rank increases and the weight for the top 20 terms drops even faster. This confirms our intuition that a small number of terms are much more important than others in a document. One analogy is that these terms are words in the "Keyword" section of a paper, except that eSearch extracts them automatically.

3.2 Automatic Query Expansion

Only publishing a document under its top terms may degrade the quality of search results. A query on a term that is not among the top terms of a document cannot find this document. We adopt automatic query expansion [23] (also called automatic relevance feedback) to

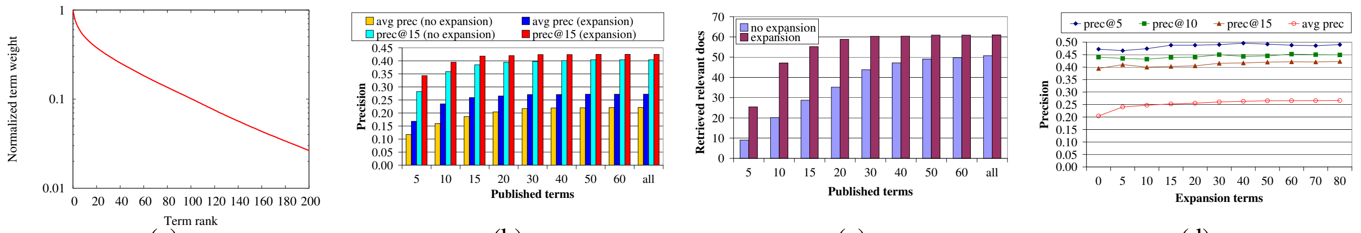

Figure 3: (a) Ranked term weight of the TREC corpus, normalized to the biggest term weight in each document. (b) eSearch's precision with respect to the number of terms under which a document is published. The performance of the "all" series is equivalent to that of a centralized system. (c) The average number of retrieved relevant documents for a query when returning 1,000 documents. (d) eSearch's precision with respect to the number of expanded query terms.

alleviate this problem. We draw on the observation that, with more terms in a query, it is more likely that a document relevant to this query is published under at least one of the query terms. This scheme automatically expands a short query with additional relevant terms. It has been show as one important technique to improve performance in centralized IR systems [36].

We experimented with several query expansion techniques and found that complex ones such as [23] only marginally improve search quality for the TREC corpus compared with simple ones. For the sake of clarity we describe below a simple scheme that is degenerated from [23] but has the same performance.

Given a query, eSearch first uses the hybrid indexing structure to retrieve a small number f of best matching documents. We call these documents *feedback documents*. For each term in the feedback documents, the Engine node that starts the search on behalf of a client computes the average weight of terms in the feedback documents and chooses k terms that have the biggest average weight. These terms are assumed to be relevant to the query and are added into the query. The new query is then used to retrieve the final set of documents for return. Recall that VSM assigns a weight for query terms as it does for document terms (see Equation 1). The weight for an expanded query term, which is not assigned by VSM, is its average weight in feedback documents divided by a constant α. Through experiments we found that $f = 10$ and $\alpha = 16$ turned out to work well. Alternatively, one may set α to a very large number to make the overall ranking equivalent to that without the expanded query terms, but still search nodes corresponding to expanded terms to reduce the chance of missing relevant documents.

We illustrate these steps through an example. Suppose VSM identifies "routing" as the only important term for a document D. Given a query of "computer network", eSearch first retrieves relevant documents with either "computer" or "network" as one of their important terms. After a look at the retrieved documents, eSearch finds that "routing" seems to be an important common word among them. It then expands the query as "computer network routing", assigning "routing" a relatively smaller weight than the weight for the original query terms "computer network". The new query is then used to retrieve a final set of documents. This time, it can find document D.

3.3 Experimental Results

We experiment with the TREC corpus to determine proper parameters for eSearch, including the number of top terms under which a document is published and the number of expanded terms for a query. We use the "title" field of TREC topics 351-450 as queries. On average, each query consists of 2.4 terms and has 94 manually identified relevant documents in the corpus. Note that this identification is subjective and based on user input, which implies that a document that contains all terms in a query is not necessarily relevant to the query, and a document relevant to a query need not contain all terms in the query.

The metric to quantify the quality of search results is *precision*, defined as the number of retrieved relevant documents divided by the number r of retrieved documents. For instance, $prec@15$=0.4 means that, when returning 15 documents for a query, about 40% of the returned documents will be evaluated by users as really relevant to the query. $prec@r$ varies with r. The *average precision* for a single topic is the mean of the precision obtained after each relevant document is retrieved (using zero as the precision for relevant documents that are not retrieved). We are particularly interested in high-end precision (e.g., $prec@15$) as users usually only view the top 10 search results.

Figure 3 (b) reports eSearch's precision with respect to the number of terms under which a document is published (shown on the X axis). The "expansion" and "no expansion" series are with and without query expansion, respectively. In this experiment, we set the number of expanded terms to 30. "All" means that a document is published under all its terms, whose precision is equal to that of a centralized IR system. Two observations can be drawn from this figure. (1) eSearch can approach the precision of the centralized baseline by publishing a document under a small number of selected terms, e.g., top 20 terms. (2) Automatic query expansion improves precision, particularly when documents are published under very few top terms.

We next examine the performance when returning a large number of documents for each query. This is the case when a user wishes to retrieve as many relevant documents as possible. Figure 3 (c) reports the average number of retrieved relevant documents for a query when returning 1,000 documents. Query expansion increases the number of retrieved relevant documents by 22%. The performance of the "expansion" series catches up with that of the centralized baseline earlier than the "no expansion" series, showing that the use of query expansion allows eSearch to publish documents under fewer terms to achieve the performance of centralized systems.

Figure 3 (d) shows the precision with respect to the number of expanded query terms. In this experiment, each document is published under its top 20 terms. The high-end precision is not very sensitive to query expansion. The average precision improves slowly after the number of expanded terms exceeds 10. Adding more terms into a query results in searching more nodes. This figure indicates that the benefit of query expansion can be reaped with limited overhead in eSearch.

Comparing the "no expansion" series in Figures 3 (b) and (c), we find that top search results are relatively insensitive to top term selection whereas the low-rank search results are affected more severely. Accordingly, query expansion is most useful for improving low-end precision but not for high-end precision. This observation leads to a further optimization. Given a new query, eSearch first retrieves a small number of documents without query expansion. If the user is unsatisfied with the results, eSearch uses these documents as feedback documents to expand the query and do a second round of search to return more documents.

Figures 3 (b) to (d) show that eSearch's average retrieval quality is as good as the centralized baseline. In Table 1 we try to further understand the difference for individual queries between eSearch and the baseline. When documents are published under a few terms, eSearch performs badly for a few queries. For instance, when publishing documents under only their top 5 terms, there is one query that eSearch finds no relevant documents for whereas the baseline can find 9 relevant documents. When the number of selected top terms increases, eSearch's performance improves quickly. When publishing documents under their top 20 terms, the worst relative performance for eSearch is one query for which eSearch retrieves four fewer relevant documents.

difference	top 5 terms	top 10 terms	top 20 terms
d=3	1	0	0
d=2	3	2	0
d=1	5	3	3
d=0	53	73	90
d=-1	15	9	5
d=-2	6	8	0
d=-3	4	3	1
d=-4	3	0	1
d=-5	3	1	0
d=-6	3	0	0
d=-7	3	1	0
d=-8	0	0	0
d=-9	1	0	0

Table 1: Difference in the number of retrieved relevant documents between eSearch and the centralized baseline when returning 10 documents for each query. The columns correspond to eSearch with different configurations (without using query expansion). One entry with value p in the "d=k" row (e.g., the entry with value 8 in the "d=-2" row) means that, out of the 100 TREC queries, p queries return k more relevant documents in eSearch than in the baseline (or return fewer relevant documents if $k<0$). For instance, when publishing documents under their top 10 terms, for 8 queries, eSearch returns 2 fewer relevant documents than the baseline, and for 73 queries, eSearch performs the same as the baseline. For some queries, eSearch does better than the baseline because of the inherent fuzziness in Okapi's ranking function (i.e., just focusing on important terms may actually improve retrieval quality sometimes).

3.4 Discussions

Our evaluation so far has assumed that eSearch always publishes documents under the same number of top terms. This number can actually be varied from document to document. Intuitively, it may be more reasonable to publish long documents under more terms. Our current implementation includes a heuristic that publishes documents with big term weights under more terms. A detailed discussion is omitted due to space limitations.

The worst case scenario for eSearch is that a query is about a term that is not important in *any* document, e.g., the phrase "because of". eSearch cannot return any result although documents containing this term do exist. Although this scenario does exist in theory, in practice, eSearch's search quality on the widely used TREC benchmark corpus is as good as the centralized baseline. To achieve good retrieval quality, the absolute number of selected top terms may vary from corpus to corpus, but we expect it to be a small percentage of the total number of unique terms in a document, as the importance of terms in a document decreases quickly (see Figure 3 (a)). The same trend also holds for several other copora we tested, including MED, ADI, and CRAN (available in the SMART package [5]). Moreover, it is always possible to flood a hard query to every node, degrading eSearch's efficiency to Gnutella in really rare cases.

It should be emphasized that eSearch is not tied to any particular document ranking algorithm. For documents with cross reference links, link analysis algorithms such as Google's PageRank [4] can be incorporated, for instance, by combining PageRank with VSM to assign term weights [21] or publishing important Web pages (identified by PageRank) under more terms.

4 Disseminating Document Metadata

Given a new document, eSearch uses Okapi to identify its top terms and distributes its term list to nodes responsible for those terms. Since the same data are sent to multiple recipients, one natural way to economize on bandwidth is to multicast the data to the recipients.

Due to a variety of deployment issues, IP multicast is not widely supported in the Internet. Instead, several overlay multicast systems have been recently proposed, e.g., Narada [9]. These systems usually target multimedia or content distribution applications, where a multicast session is long. Thus, they can amortize the overhead for network measurement, multicast tree creation, adaptation, and destruction, over a long data session.

The scenario for term list dissemination, however, is quite different. Typically, a document is only several kilobytes and its term list is even smaller. As a result, eSearch disseminates data through a large number of extremely short sessions. Although the absolute saving from multicast in a single session is small, the aggregate savings over a large number of sessions would be huge. But this requires protocol overhead for multicast tree creation and destruction to be very small.

To this end, we propose overlay *source* multicast to distribute term lists. It does not use costly messaging to explicitly construct or destroy the multicast tree. Assisted by Internet distance estimation techniques such as GNP [25], the data source locally computes the structure of an efficient multicast tree that has itself as the root and includes all recipients. It builds an application-level packet with the data to be disseminated as payload and the structural information of the multicast tree as header, which specifies the IP addresses of the recipients and the parent-child relationship among them. It then sends the packet to the first-level children in the multicast tree. A recipient of this packet inspects the header to find its children in the multicast tree, strips off information for itself from the header, and forwards the rest of the packet to its children, and so forth.

In the following, we provide more details on the method that the data source uses to compute the structure of the multicast tree. Our method is based on GNP. A node using GNP measures RTTs to a set of well-known landmark nodes and computes coordinates from a high-dimensional Cartesian space for itself. Internet distance between two nodes is estimated as the Euclidean distance between their coordinates.

When a node S wishes to send a term list to nodes responsible for top terms in a document, it performs concurrent DHT lookups to locate all recipients. Each recipient directly replies to node S with its IP address and its GNP coordinates. After hearing from all recipients, node S locally builds a fully connected graph (a clique) with itself and recipients as vertices. It annotates each edge with estimated Internet distance, i.e., the Euclidean distance between the coordinates of the two nodes incident with the edge. In practice, other factors such as bandwidth and load could also be considered when assigning weights to edges. Finally, it runs a minimum spanning tree algorithm over the graph to find an efficient multicast tree.

The lookup results for recipients' IP and GNP coordinates are cached and reused for a certain amount of time. Stale information used before it times out will be detected when a node receives a term list that it should not be responsible for, which will trigger a recovery mechanism to find the correct recipient.

We use simulations to quantify the savings from overlay source multicast. Three data sets are used in our experiments. The first one is derived from a 1,000 node transit-stub graph generated by GT-ITM [6]. We use its default edge weight as the link latency. The second one is derived from NLANR's one-day RTT measurements among 131 sites scattered in the US [26]. After filtering out some unreachable sites, we are left with 105 fully connected sites and the latency between each pair of them. We use the median of one-day measurements as the end-to-end latency. The third one is an Internet Autonomous System (AS) snapshot taken by the Route Views Project [27] on April 28, 2003, with a total of 15,253 AS'es recorded. Since the latency between adjacent AS'es is unknown, we assign latency randomly between 8ms and 12ms. For the GT-ITM and AS data set, we randomly assign some user nodes to routers or AS'es. The end-to-end latency between two nodes is computed as the latency of the shortest path between them.

Figure 4 reports the performance of overlay source multicast using these data sets. In all experiments, we use the k-means clustering algorithm to select 15 center nodes as landmarks and then use GNP's technique to compute node coordinates from a 7-dimensional Cartesian space. For each data set, we randomly choose some nodes (shown on the X axis) as recipients to join the mul-

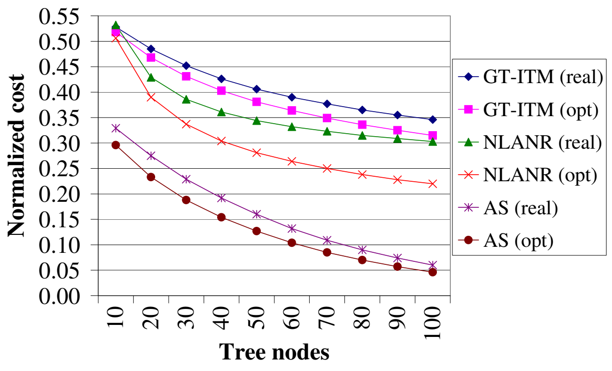

Figure 4: Cost of overlay source multicast normalized to that of using separate unicast to deliver data.

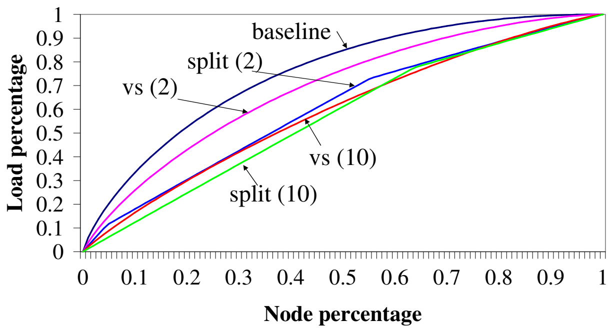

Figure 5: Comparison of load balancing techniques using equal-size objects. "baseline" is the basic Chord without virtual server. "vs (k)" is a Chord with k virtual nodes running on each physical node. "split (k)" is our technique where a new node performs k random lookups and splits the overloaded node.

ticast tree. The cost of a multicast tree is the sum of the cost of all edges. The Y axis is the multicast tree's cost normalized to that of using separate unicast to deliver the term list to recipients. For each data set, the "opt" curve is the normalized cost of the optimal minimum spanning tree assuming that the real latency between each pair of nodes is known. The "real" curve (result of our scheme) is the normalized cost of the minimum spanning tree constructed using estimated Internet distance.

As can be seen from the figure, with 20 nodes in the tree, multicast reduces the communication cost by up to 72%. In all cases, the performance of the "real" curve approaches that of the "opt" curve, indicating that GNP estimates Internet distance with reasonable precision. The performance gap between the "opt" and "real" curve for the NLANR data set widens as the number of tree nodes increases. This is because GNP is not very accurate at estimating short distances. Some of NLANR's monitoring sites are very close to each other, particular at the east coast. This data set has many RTTs under 10ms. With more nodes added to the multicast tree, there is a bigger chance that some of them are very close to each other. Accordingly, GNP's estimation error increases and the resulting minimum spanning tree is less optimal.

5 Balancing Term List Distribution

One problem not addressed in previous studies on P2P keyword search [16, 20, 28, 39] is the balance of metadata distribution across nodes. Because popularity of terms varies dramatically, nodes responsible for popular terms will store much more data than others. A traditional approach to balance load in DHTs is *virtual server* [38], where a physical node functions as several virtual nodes to join the P2P network. As a result, the number of routing neighbors that a physical node monitors increases proportionally. More importantly, we find that virtual server is incapable of balancing load when the length of inverted lists varies dramatically.

Our solution is a combination of two techniques. First, we slightly modify Chord's node join protocol. A new node performs lookups for several random keys. Among nodes responsible for these keys, it chooses one that stores the largest amount of data and takes over some of its data. Second, each term is hashed into a key range rather than a single key. For an unpopular term, its key range is mapped to a single node. For a popular term, its key range may be partitioned among multiple nodes, which collectively store the inverted list for this term. Our experiments show that these two techniques can effectively balance the load even if the length of inverted lists varies dramatically. In addition, they avoid the extra maintenance overhead that virtual server introduces.

5.1 Node Join Protocol

In Chord, a new node performs a lookup for a random key K_r and splits the key range of the node n that is responsible for this key. Originally, node n is responsible for key range $[K_n^b, K_n^e]$. After the split, the new node and node n are responsible for key ranges $[K_n^b, K_r]$ and $[K_r + 1, K_n^e]$, respectively. With virtual server, the new node functions as several virtual nodes. Each virtual node executes this join protocol once.

Instead of splitting the key range of a random node, our technique seeks to split the key range of an overloaded node. In eSearch, a new node performs lookups for k random keys. Among nodes responsible for these keys, it chooses one that stores the largest amount of data to split. If nodes have heterogeneous storage capacity, it chooses the one that has the highest relative load to split.

Figure 5 compares the load balancing techniques. In this experiment, we distribute one million equal-size objects to a 10,000-node Chord. Nodes are sorted in decreasing order according to the number of objects they

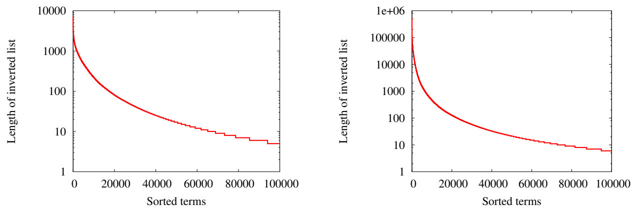

Figure 6: Length distribution of the TREC corpus's inverted lists when documents are published under (a) top 20 terms, or (b) all terms.

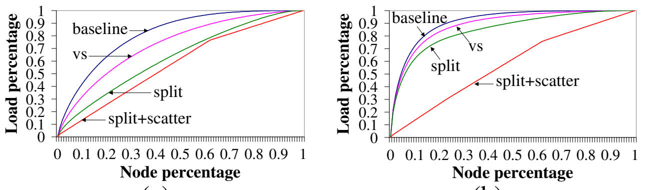

Figure 7: Comparison of load balancing schemes using (a) the "top 20 terms" load, or (b) the "all terms" load.

store. The X-axis shows the percentage of the total number of nodes in the system for which the Y-axis gives the percentage of objects hosted by the corresponding nodes. "baseline" is the basic Chord without virtual server. "vs (k)" is a Chord with k virtual nodes running on each physical node. "split (k)" is our technique in which a new node performs k random lookups and splits the overloaded node. The closer the graph is to being linear, the more evenly distributed the load. This figure shows that our technique balances load better than virtual server, particularly when k is small. Note that this benefit is achieved without virtual server's maintenance overhead.

5.2 Distributing Load for a Single Term

As the popularity of terms varies dramatically, the length of inverted lists also does. Figure 6 plots the length distribution of TREC's inverted lists when documents are published under top 20 terms or all terms. In the following, we will simply refer to them as the "top 20 terms" load or the "all terms" load. Note that the Y axis is in log scale, and the numbers in Figure 6 (b) are larger than that in Figure 6 (a) by two orders of magnitude. The length distribution is skewed for both cases, particularly for the "all terms" load. Only publishing documents under the top 20 terms greatly reduces the length of long inverted lists corresponding to popular terms since they are actually not important in many documents they appear in.

Because of this variation in the length of inverted lists, the load balancing techniques in Section 5.1 cannot work well on their own. Our complementary technique is to hash each term t into a key range $[K_t^b, K_t^e]$ rather than a single key. The key range of an unpopular term is mapped to a single node whereas the key range of a popular term may be mapped to multiple nodes, which collectively store the inverted list of this term.

Chord uses 160-bit keys. We partition keys into two parts: 140 high-order bits and 20 low-order bits. Given a document D, we first identify its top terms. For each top term t, we generate a key K_t for it. The high-order bits K_t^h of key K_t are the high-order bits of the SHA-1 hashing [24] of the term's text. Let $K_t^b = 2^{20}K_t^h$. The low-order bits of K_t are generated randomly. We then store document D's term list on the node that is responsible for key K_t. As a result, the term lists for documents that have term t as one of their important terms will be stored in key range $[K_t^b,\ K_t^b + 2^{20} - 1]$. If this key range is partitioned among multiple nodes, then the inverted list for term t is partitioned among these nodes automatically.

The search process needs a change. A query containing term t is first routed to the node responsible for key K_t^b and then forwarded along Chord's ring until it reaches the node responsible for key K_t^b+2^{20}-1. All nodes in this range participate in processing this query since they hold part of the inverted list for term t.

The node join protocol is also changed slightly. When a new node arrives, it obtains a random document (through any means) and randomly chooses k terms that are not stopwords from the document. For each chosen term t, it uses the process described above to compute a key K_t for it. Among nodes responsible for these generated keys, it chooses one that stores the largest amount of data to split. The reason why we use random terms from a random document to generate the bootstrapping keys is to force the distribution of these keys to follow the inverted list distribution such that long inverted lists are split with a higher probability.

No other change to Chord is needed. Importantly, there is no specific node that maintains a list of nodes that store the inverted list of a given term, i.e., our technique is completely decentralized. Any node joining the partition of a term can fail independently. Node failure is handled by Chord's default protocol.

While the above describes the node join protocol in order to balance load in the presence of active joins, a similar protocol may be used by stable nodes to periodically redistribute load in the absence of joins. We do not implement or evaluate this optimization in this paper.

5.3 Experimental Results

Figure 7 compares the load balancing techniques. The "baseline", "vs', and "split" curves are the same as those

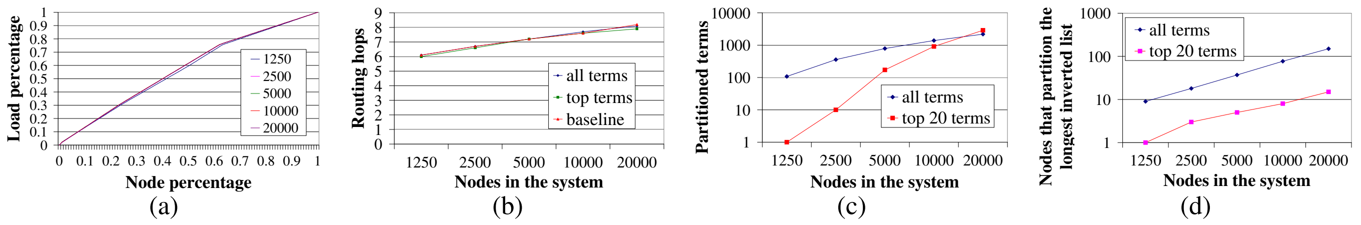

Figure 8: Scalability of our load balancing technique ("split+scatter"). (a) Metadata distribution. (b) Average routing hops. (c) Number of partitioned terms. (d) Number of nodes that partition the longest inverted list.

in Figure 5. "split+scatter" is our complete load balancing scheme, mapping a term into a key range and splitting overloaded nodes. For virtual server ("vs"), each physical node functions as 10 virtual nodes. For our techniques ("split' and "split+scatter"), a new node selects an overloaded node from 10 random nodes to split. We distribute the "top 20 terms" load and the "all terms" load to a 10,000-node Chord, respectively. Although the "all terms" load is not how eSearch actually works, we choose it to represent a scenario where the length distribution of eSearch's inverted lists becomes extremely skewed because, for instance, term lists for a gigantic number of documents are stored in eSearch.

In Figure 7, the load for the "baseline" is unbalanced, due to the large variation in length of inverted lists. For the "all terms" load, 1% of the nodes store 21.5% of the term lists. Virtual server improves the situation marginally—for the "all terms" load, 1% of the nodes still store 20.3% of the term lists. Splitting overloaded nodes ("split") performs better than virtual server ("vs") but only our complete technique ("split+scatter") is able to balance the load when the inverted lists are extremely skewed, owing to its ability to partition the inverted list of a single term among multiple nodes.

The rest of our experiments evaluate the scalability of "split+scatter", using the "all terms" load and varying the number of nodes in the system from 1,250 to 20,000. The load distribution is reported in Figure 8 (a). The curves overlap, indicating that "split+scatter" scales well with the system size. Figure 8 (b) shows the average routing hops in Chord with different configurations. "baseline" is the default Chord. The other two curves are "split+scatter" with different loads. All curves overlap, indicating that our modifications to Chord do not adversely affect Chord's routing performance.

Figure 8 (c) reports the number of terms whose inverted lists are stored on more than one node. Note that both the X axis and the Y axis are in log scale. As the number of nodes increases, the number of partitioned terms increases proportionally. The curve for the "top 20 terms" load grows faster. Because its inverted lists are not extremely skewed, more added nodes are devoted to partition unpartitioned terms, whereas in the "all terms" load added nodes are mainly used to repetitively partition terms with extremely long inverted lists. Figure 8 (d) reports the number of nodes that collectively host the longest inverted list. As the number of nodes increases, this inverted list is partitioned among more nodes proportionally. The partition in the "all terms" load is higher than that in the "top 20 terms" load because the inverted list in the "all terms" load is much longer.

Overall, our load balance technique scales well with the system size. It balances term list distribution well under different system sizes and does not affect Chord's routing performance. As the system size increases, long inverted lists are automatically partitioned among more nodes. Since it works for two quite different loads (particularly the extreme "all terms" load), we expect it to also scale well with the corpus size.

6 Analysis of System Resource Usage

In this section, we analyze eSearch's system resource usage when publishing metadata for a document or processing a query, and compare it with P2P systems based on global indexing (so-called *Global-P2P systems*) [16, 20, 28, 39]. We don't claim the default values used in the analysis are representative for all situations. Instead, we just want to give a flavor of eSearch's resource usage.

6.1 Publishing a Document

eSearch executes a two-phase protocol to publish a document. In the first phase, it uses DHT routing to locate the nodes responsible for top terms in the document and obtain their IP address and GNP coordinates. In the second phase, it uses overlay source multicast to deliver the document. The cost for the first phase could be avoided if the needed information has already been cached locally. We assume that this cache is disabled in the following analysis.

Data transmitted to locate the recipients are $B_l = n_t * h * m_l = 10,400$ bytes, where n_t=20 is the number of top terms, h=8 is the average number of routing hops in a 20,000-node Chord [1], m_l=65 is the size of the message (including 40-byte TCP/IP headers, 1-byte identifier that specifies the type of the message, 4-byte IP address of the data source, and a 20-byte DHT key). Data replied from the recipients are $B_r = n_t * m_r$=1,220 bytes, where m_r=61 is the size of the reply message (including 28-byte UDP/IP headers [2], 1-byte message identifier, 4-byte recipient IP address, and 28-byte GNP coordinates in a 7-dimensional Cartesian space). The total data transmitted in the first phase is $B_l + B_r$=11,620 bytes.

In the second phase, there are two options for the content to be multicast to the recipients. The data source can build the term list for the document and multicast the term list. Alternatively, the data source can multicast the document itself, leaving it to the recipients to build the term list. The first method is more efficient in that a term list is smaller than a document, but the second method allows more flexible retrieval. With document text in hand, eSearch can search exact matches for quoted text, provide sentence context for matching terms, and support a "cached documents" feature similar to that in Google. We opt for multicasting the document itself.

Using statistics from the TREC corpus, we assume that the average document length is 3,778 bytes and it can be compressed to 1,350 bytes (a 2 to 4 text compression ratio is typical for bzip2). The UDP/IP headers, message identifier, and structural information of the multicast tree add 150 bytes to the multicast message, resulting in a 1,500-byte packet. Based on the simulation results in Section 4, we conservatively estimate that multicast can save bandwidth by 55%. Thus it consumes 20*1500*0.45=13,500 bytes bandwidth to multicast a document to 20 recipients. The total (phase one and phase two) cost for publishing a document is 11620+13500=25,120 bytes.

Next, we calculate the bandwidth consumption to distribute metadata for a document in a Global-P2P system. Although these systems [16, 20, 28, 39] did not propose using stopword removal, we add in this step. It significantly reduces the size of the metadata since as much as 50% of a document's content is stopwords. According to the TREC corpus, each document on average contains 153 unique terms after stemming and stopword removal. The metadata for a term in a document includes the term ID, the document ID, and a 1-byte attribute specifying, for instance, the frequency of the term in the document. (If this information needs more than one byte, approximating it with a 256-level value would provide sufficient precision.) The term ID is a DHT key of 20 bytes. We assume the document ID is 8 bytes, including the IP address of the node that stores the document, the port number through which to establish a connection with that node, and a 2-byte document number that differentiates documents on that node.

In a Global-P2P system, data transmitted to publish a document are $B = n_a * h * m = 80,784$ bytes, where n_a=153 is the number of terms in the document, h=8 is the routing hops in Chord, m=66 is the size of the message (including 40-byte TCP/IP headers, 1-byte message identifier, 16-byte term ID, 8-byte document ID, and 1-byte attribute). This bandwidth consumption is 3.2 times that of eSearch. This inefficiency is due to routing high-overhead small packets in the overlay network. Suppose the overhead of overlay routing can be reduced from h=8 to approximately h=2 by introducing proximity neighbor selection into Chord [18]. Then a Global-P2P system consumes 20,196 bytes bandwidth to publish a document whereas eSearch consumes 17,320 bytes bandwidth.

Global-P2P systems send a large number of small messages to publish a document. eSearch, in contrast, sends a small number of large messages (the whole document). If the inefficiency of processing a large number of small messages in routers and end hosts is counted in, savings in eSearch would be even more significant. Moreover, eSearch distributes the actual document, allowing more flexible retrieval.

6.2 Processing a Query

When a user intends to retrieve a small number of best matching documents, by default, query expansion is not used in eSearch (please refer to discussion in Section 3.3). The bandwidth cost to process a query is $B_q = n_q * h * m_q + n_q * m_d = 3,335$ bytes, where $n_q = 5$ is the number of nodes responsible for the query terms, $h = 8$ is the routing hops in the Chord, $m_q = 61$ is the size of the query message (including 40-byte TCP/IP headers, 1-byte message identifier, and 20-byte query text), and $m_d = 28 + 1 + 15 * (8 + 2) = 179$ is the size of the search results (including 28-byte UDP/IP headers, 1-byte message identifier, and 8-byte ID and 2-byte relevance score for 15 matching documents). Both query and publishing costs are independent of the size of the corpus and grow slowly (logarithmically) with the number of nodes in the system. In contrast, a local indexing system sends a query to every node in the system whereas the cost to process multi-term queries in a global indexing system grows with the size of the corpus.

Occasionally, the user is not satisfied with the search results and requests a feedback process to retrieve a larger number of documents using query expansion. The node that started the search first collects the metadata of feedback documents in order to select terms to be added into the query. The bandwidth cost is $B_c = n_f * m_f =$

[1]The actual delay stretch in a proximity-aware overlay [18] may be smaller than the hop counts. This effect is discussed later.

[2]Throughout the analysis, we assume that communication between routing neighbors uses pre-established TCP connections whereas short communication between non-neighboring nodes uses UDP.

$10,000$ bytes, where $n_f = 10$ is the number of feedback documents and $m_f = 1,000$ bytes is the cost to retrieve metadata for one feedback document. It then starts a second round of search using the expanded query. The analysis of bandwidth consumption is similar to that in the first round, $B_q = n_q * h * m_q + n_q * m_d = 315,510$ bytes, except that it searches 30 nodes ($n_q = 30$) and each node returns 1,000 documents ($m_d = 28+1+1000*(8+2) = 10,029$). In total, the feedback round consumes 325,510 bytes bandwidth, the majority of which is due to returning a large number of documents.

6.3 Storage Cost

The term list for a document is replicated about 20 times in eSearch, but its storage cost is not 20 times that of the Global-P2P systems, as explained below. To speed up query processing, each eSearch node locally builds inverted lists for documents that it holds term lists for. It also maintains a table that maps a document's 8-byte global ID into a 2-byte local ID. Although further compression is possible [43], we assume that 3 bytes are used for each (non-stopword) term in a document, 2 bytes for the local document ID, and 1 byte for an attribute (e.g., the frequency of the term in the document). The total cost to store metadata for a document in eSearch (including the cost for the mapping table) is 20*(8+2+153*(2+1))=9,380 bytes, assuming each document has 153 terms after stemming and stopword removal. Since storing full document text is an additional feature, we do not count it in the comparison.

Global-P2P systems need at least 9 bytes for a term in a document, 8 bytes for the global document ID and 1 byte for the attribute. The total storage cost for a document's metadata is 153*(8+1)=1,377 bytes. In these systems, information for terms in a document is distributed on different nodes. They cannot benefit from the technique that maps a document's long global ID into a short local ID since the size of one entry of the mapping table already exceeds the size of the information for a term and is not reused sufficiently to justify the cost.

The storage space consumed by eSearch is 6.8 times that of the Global-P2P systems. The benefit is its low search cost. According to Blake and Rodrigues [2], disk capacity has increased 160 times faster than the network bandwidth for an end user. Therefore, we believe trading modest disk space for communication and precision is a proper design choice for P2P systems. We also plan to adopt index compression [43] and pruning [7] to further reduce storage consumption.

7 Related Work

Compared with our P2P architecture, distributed IR systems such as GlOSS [17] uses a hierarchy of metadatabases to summarize contents of other databases. During a search, the summary is referenced to choose databases that may contain most relevant documents. We use semantic information produced by VSM to guide term list replication. Carmel et al. [7] used similar information to guide index pruning in a centralized site.

Below, we classify recently proposed P2P search systems into three categories according to the type of network in which they operate.

Search in Distributed Hash Table Systems

Global indexing based P2P keyword search systems built on top of DHTs are most relevant to eSearch. To answer multi-term queries, these systems must transmit inverted lists over the network to perform a join. Several techniques have been proposed to reduce this cost.

In KSS [16], the system precomputes and stores results for all possible queries consisting of up to a certain number of terms. The number of possible queries unfortunately grows exponentially with the number of terms in the vocabulary. Reynolds and Vahdat [28] adopted a technique developed in the database community to perform the join more efficiently. This technique transmits the Bloom filter of inverted lists instead of the inverted lists themselves. Suel et al. [39] adopted Fagin's algorithm to compute the top-k results without transmitting the entire inverted lists. This algorithm transmits the inverted lists incrementally and terminates early if sufficient results have already been obtained. Li et al. [20] suggested combining several techniques to reduce the cost of a distributed join, including caching, Bloom filter, document clustering, etc.

These approaches are orthogonal to our efforts in eSearch. Our hybrid indexing architecture intends to completely eliminate the cost for distributed join. A quantitative comparison of the search cost between them and eSearch would be an interesting subject of future work. Even with various optimizations, we still expect their cost to grow with the corpus size, perhaps at a rate slower than that of a basic global indexing system.

Search in Unstructured Peer-to-Peer Networks

Centralized indexing systems such as Napster suffer from a single point of failure and bottlenecks at the index server. Flooding-based techniques such as Gnutella send a query to every node in the system, consuming huge amounts of network bandwidth and CPU cycles. To reduce the search cost, heuristic-based approaches try to direct a search to only a fraction of the node population.

Rhea and Kubiatowicz [29] described a method in which each node uses Bloom filters to summarize its neighbors' content. A query is only forwarded to neighbors that may have relevant documents with a high probability. PlanetP [13] uses Bloom filters to summarize content on each node and floods the summary to the entire system. Crespo and Garcia-Molina [12] introduced the notion of Routing Indices that give a promising "direction" toward relevant documents.

Replication has also been explored to improve search efficiency. FastTrack [15] designates high-bandwidth nodes as super-nodes. Each super-node replicates the indices of several other nodes. Cohen et al. [11] found that setting the number of object replicas to the square root of the searching rate for an object minimizes the expected search size on successful queries.

Lv et al. [22] found that random walk and expanding-ring search are more efficient than flooding. Chawathe et al. [8] combined several techniques, including random walk, topology adaption, replication, and flow control, to improve Gnutella.

Search in Networks with Semantic Locality

Schwartz [35] described a method that organizes nodes with similar content into a group. A search starts with random walk but proceeds more deterministically once it hits in a group with matching content. SETS [1] arranges nodes into a topic-segmented overlay topology where links connect nodes with similar content. Motivated by research in data mining, Cohen et al. [10] used guide-rules to organize nodes satisfying certain predicates into an associative network. Sripanidkulchai et al. [37] extended an existing P2P network by linking a node to other nodes that satisfied previous queries.

Unlike the above systems, pSearch [40] is a P2P IR system that employs statistically derived conceptual indices instead of keywords for retrieval. pSearch uses latent semantic indexing (LSI) to guide content placement in a Content-Addressable Network (CAN) such that documents relevant to a query are likely be colocated on a small number of nodes. During a search, both pSearch and eSearch transmit a small amount of data and search a small number of nodes, but eSearch is relatively more efficient if compared quantitatively. In pSearch, the problem of efficiently deriving the conceptual representation for a large corpus is also very challenging. pSearch's infrastructure, however, supports content-based retrieval of multimedia data such as image or music files.

8 Conclusion

In this paper, we proposed a new architecture for P2P information retrieval, along with various optimization techniques to improve system efficiency and the quality of search results. We made the following contributions:

- Challenging conventional wisdom that uses either local or global indexing, we proposed hybrid indexing that employs selective term list replication to combine the benefits of local and global indexing while avoiding their limitations.
- We used semantic information provided by modern IR algorithms to guide the replication. The term list of a document is only replicated on nodes corresponding to important terms in the document.
- We adopted automatic query expansion in a P2P environment to alleviate precision degradation introduced by selective replication.
- We devised a novel overlay *source* multicast protocol that has very low protocol overhead in order to reduce term list dissemination cost.
- We introduced two techniques to balance term list distribution to a greater degree than is achievable using existing load balancing techniques while avoiding their maintenance overhead.

We have quantified the efficiency of eSearch (in terms of bandwidth consumption and storage cost) and the quality of its search results by experimenting with one of the largest benchmark corpora available in the public domain. Our results show that the combination of our proposed techniques results in a system that is scalable and efficient, and achieves search results as good as a centralized baseline that uses Okapi.

Our future work includes studying eSearch's sensitivity to the global statistics produced from samples, incorporating a P2P implementation [34] of Google's PageRank algorithm, and implementing index compression [43] and pruning [7] to reduce storage consumption. We also plan to experiment with a large HTML corpus crawled from the Web.

Acknowledgments

This work is related to the pSearch project [40] that Chunqiang Tang worked on during his internship at HP Labs in year 2002. We thank Mallik Mahalingam and Zhichen Xu for their contributions in the pSearch project; Gautam Altekar, Wenrui Zhao, the anonymous reviewers, and our shepherd Robert Morris for their valuable feedback; and Liudvikas Bukys and James Roche for their support in maintaining the experimental platform. This work was supported in part by NSF grants CCR-9988361, CCR-0219848, ECS-0225413, and EIA-0080124; by DARPA/ITO under AFRL contract F29601-00-K-0182; by the U.S. Dept. of Energy Office of Inertial Confinement Fusion under Cooperative Agreement No. DE-FC03-92SF19460; and by equipment or financial grants from Compaq, IBM, Intel, and Sun.

References

[1] M. Bawa, G. S. Manku, and P. Raghavan. SETS: Search Enhanced by Topic Segmentation. In *SIGIR'03*, 2003.

[2] C. Blake and R. Rodrigues. High Availability, Scalable Storage, Dynamic Peer Networks: Pick Two. In *HotOS'03*, May 2003.

[3] W. J. Bolosky, J. R. Douceur, D. Ely, and M. Theimer. Feasibility of a Serverless Distributed File System Deployed on an Existing Set of Desktop PCs. In *SIGMETRICS'00*, 2000.

[4] S. Brin and L. Page. The Anatomy of a Large-Scale Hypertextual Web Search Engine. *Computer Networks and ISDN Systems*, 30(1–7):107–117, 1998.

[5] C. Buckley. Implementation of the SMART information retrieval system. Technical Report TR85-686, Department of Computer Science, Cornell University, Ithaca, NY 14853, May 1985. Source code available at ftp://ftp.cs.cornell.edu/pub/smart.

[6] K. Calvert, M. Doar, and E. W. Zegura. Modeling Internet Topology. *IEEE Communications Magazine*, June 1997.

[7] D. Carmel, D. Cohen, R. Fagin, E. Farchi, M. Herscovicl, Y. S. Marrek, and A. Scoffer. Static Index Pruning for Information Retrieval Systems. In *SIGIR'01*, 2001.

[8] Y. Chawathe, S. Ratnasamy, L. Breslau, N. Lanham, and S. Shenker. Making Gnutella-like P2P Systems Scalable. In *SIGCOMM'03*, 2003.

[9] Y. Chu, S. G. Rao, and H. Zhang. A Case for End System Multicast. In *SIGMETRICS'00*, 2000.

[10] E. Cohen, A. Fiat, and H. Kaplan. Associative Search in Peer to Peer Networks: Harnessing Latent Semantics. In *INFOCOM'03*, April 2003.

[11] E. Cohen and S. Shenker. Replication Strategies in Unstructured Peer-to-Peer Networks. In *SIGCOMM'02*, 2002.

[12] A. Crespo and H. García-Molina. Routing Indices for Peer-to-peer Systems. In *ICDCS'02*, July 2002.

[13] F. M. Cuenca-Acuna, C. Peery, R. P. Martin, and T. D. Nguyen. PlanetP: Using Gossiping to Build Content Addressable Peer-to-Peer Information Sharing Communities. In *the 12th IEEE International Symposium on High Performance Distributed Computing (HPDC'03)*, June 2003.

[14] F. Dabek, M. Kaashoek, D. Karger, R. Morris, and I. Stoica. Wide-area cooperative storage with CFS. In *SOSP'01*, October 2001.

[15] FastTrack Peer-to-Peer technology company, 2001. http://www.fasttrack.nu.

[16] O. D. Gnawali. A Keyword Set Search System for Peer-to-Peer Networks. Master's thesis, Massachusetts Institute of Technology, June 2002.

[17] L. Gravano, H. García-Molina, and A. Tomasic. GlOSS: text-source discovery over the Internet. *ACM Transactions on Database Systems*, 24(2), 1999.

[18] K. P. Gummadi, R. Gummadi, S. D. Gribble, S. Ratnasamy, S. Shenker, and I. Stoica. The Impact of DHT Routing Geometry on Resilience and Proximity. In *SIGCOMM'03*, August 2003.

[19] IRIS. http://www.project-iris.net.

[20] J. Li, B. T. Loo, J. Hellerstein, F. Kaashoek, D. R. Karger, and R. Morris. On the Feasibility of Peer-to-Peer Web Indexing and Search. In *IPTPS'03*, February 2003.

[21] X. Long and T. Suel. Optimized Query Execution in Large Search Engines with Global Page Ordering. In *VLDB'03*, 2003.

[22] Q. Lv, P. Cao, E. Cohen, K. Li, and S. Shenker. Search and Replication in Unstructured Peer-to-Peer Networks. In *ICS'02*, June 2002.

[23] M. Mitra, A. Singhal, and C. Buckley. Improving Automatic Query Expansion. In *SIGIR'98*, 1998.

[24] National Institute of Standards and Technology. Secure Hash Standard, FIPS 180-1, April 1995.

[25] T. S. E. Ng and H. Zhang. Predicting Internet Network Distance with Coordinates-Based Approaches. In *INFOCOM'02*, 2002.

[26] NLANR. http://watt.nlanr.net/.

[27] Oregon Route Views Project. http://routeviews.org.

[28] P. Reynolds and A. Vahdat. Efficient Peer-to-Peer Keyword Searching. In *Middleware'03*, June 2003.

[29] S. Rhea and J. Kubiatowicz. Probabilistic Location and Routing. In *INFOCOM'02*, 2002.

[30] K. M. Risvik and R. Michelsen. Search Engines and Web Dynamics. *Computer Networks*, 39(3):289–302, 2002.

[31] S. E. Robertson, S. Walker, S. Jones, M. M. Hancock-Beaulieu, and M. Gatford. Okapi at TREC-3. In *TREC-3*, 1994.

[32] A. Rowstron and P. Druschel. Storage management and caching in PAST, a large-scale, persistent peer-to-peer storage utility. In *SOSP'01*, 2001.

[33] G. Salton, A. Wong, and C. Yang. A Vector Space Model for Information Retrieval. *Journal for the American Society for Information Retrieval*, 18(11):613–620, 1975.

[34] K. Sankaralingam, S. Sethumadhavan, and J. C. Browne. Distributed Pagerank for P2P Systems. In *the 12th IEEE International Symposium on High Performance Distributed Computing (HPDC'03)*, June 2003.

[35] M. Schwartz. A Scalable, Non-Hierarchical Resource Discovery Mechanism Based on Probabilistic Protocols. Technical Report TR CU-CS-474-90, University of Colorado, 1990.

[36] A. Singhal. Modern Information Retrieval: A Brief Overview. *IEEE Data Engineering Bulletin*, 24(4):35–43, 2001.

[37] K. Sripanidkulchai, B. Maggs, and H. Zhang. Enabling Efficient Content Location and Retrieval in Peer-to-Peer Systems by Exploiting Locality in Interests. *ACM SIGCOMM Computer Communication Review*, 32(1), January 2001.

[38] I. Stoica, R. Morris, D. Karger, M. F. Kaashoek, and H. Balakrishnan. Chord: A scalable peer-to-peer lookup service for internet applications. In *SIGCOMM'01*, 2001.

[39] T. Suel, C. Mathur, J. Wu, J. Zhang, A. Delis, M. Kharrazi, X. Long, and K. Shanmugasunderam. ODISSEA: A Peer-to-Peer Architecture for Scalable Web Search and Information Retrieval. In *WebDB'03*, June 2003.

[40] C. Tang, Z. Xu, and S. Dwarkadas. Peer-to-Peer Information Retrieval Using Self-Organizing Semantic Overlay Networks. In *SIGCOMM'03*, 2003.

[41] Text Retrieval Conference (TREC). http://trec.nist.gov.

[42] A. Tomasic and H. Garcia-Molina. Query Processing and Inverted Indices in Shared-Nothing Document Information Retrieval Systems. *VLDB Journal*, 2(3):243–275, 1993.

[43] I. H. Witten, A. Moffat, and T. C. Bell. *Managing Gigabytes: Compressing and Indexing Documents and Images*. Morgan Kaufmann, second edition, 1999.